(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
17 January 2002 (17.01.2002)

PCT

(10) International Publication Number
WO 02/03774 A2

(51) International Patent Classification: Not classified

(21) International Application Number: PCT/US01/21658

(22) International Filing Date: 10 July 2001 (10.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/614,521 12 July 2000 (12.07.2000) US

(71) Applicant *(for all designated States except US)*: **KESTREL TECHNOLOGIES, INC.** [US/US]; 4th Floor, 40 Broad Street, New York, NY 1004 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **FRECKLETON, Graham, M.** [US/US]; 155 East 31st Street, 20H, New York, NY 10018 (US). **GERMAN, Valery** [US/US]; 1877 East 12th Street, SG, Brooklyn, NY 11229 (US).

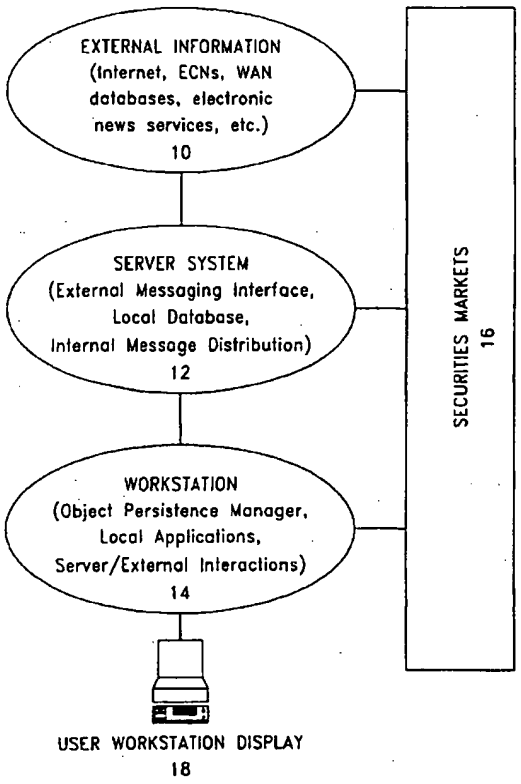
(74) Agent: **ERDMAN, Kevin, R.**; Baker & Daniels, Suite 2700, 300 N. Meridian Street, Indianapolis, IN 46204 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE,

[Continued on next page]

(54) Title: PERSISTENCE CONTROL AND COORDINATION FOR TRADING SYSTEM OBJECT ORIENTED SYSTEM AND METHOD



(57) Abstract: The present invention involves a security management and trading system and method which maintains persistent control over time sensitive data. The system interacts with data processing systems transacting the purchase and sale of at least one security having a set of characteristic data. The data processing system is operated by a plurality of trading participants and transmits information relating to the securities characteristic data. The trading system comprises a network (LAN), a workstation (14), and a server (12). The workstation is coupled to the network and includes a display (18) for presenting to a participant information about pending market conditions as they relate to the security characteristic data of at least one security. The server is in communication with the workstation over the network and provides event information to the workstation. At least one of the workstation and server includes an object oriented software program (OMI) comprising a plurality of objects. The workstation includes a securities evaluation software application (SE) which references an object having time sensitive security characteristic data. The workstation also includes persistent control software (PC) having instructions for enabling the workstation to update the object having the time sensitive security characteristic data when external event information is received and is relevant to the time sensitive securities data.

WO 02/03774 A2

IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

PERSISTENCE CONTROL AND COORDINATION FOR TRADING SYSTEM OBJECT ORIENTED SYSTEM AND METHOD

Copyright

A portion of the disclosure of this patent document contains material which is the subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent files or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

1. Field of the Invention.

The invention relates to trading system software. More specifically, the field of the invention is that of financial trading and management software for traders and investment managers of fixed income securities.

2. Description of the Related Art.

A sizable portion of investment vehicles available in today's financial markets are universally characterized as fixed income securities. Exemplary fixed income securities encompass government bonds, bills and notes auctioned at regular intervals by the U.S. and other foreign governments to finance governmental activities. This, of course, is one of many types of fixed income securities, others include corporate bonds, municipal bonds, etc. The common thread running between all fixed income securities is the payment of a set return to the investor over the life span of the security.

There are two forms of fixed income return to the investor. The first involves the provision of coupon payments at regular intervals, at the stated interest rate of the security. For example, a ten-year note may specify an 8% rate of interest on a $1,000 par value with coupons coming due twice each year for ten years. This translates to two $40 payments to the holder of the

note for ten years with a final payment of $1040 (principal and interest). The other form of bond is called a zero coupon, or discount bond which provides no payment except for the final return of the face value of the bond at a specified date (e.g. ten years from issuance). The discount bond is sold at some fraction of its face value, with the interest rate discount a function of this and the term of the bond.

The fixed income securities distributed by the United States Government are known as U.S. treasuries. These instruments span maturity terms of 13 to 52 weeks (T-bills), one to ten years (notes), and up to 30 years (bonds). The T-bills are pure discount securities having no coupons. All other treasuries having longer terms are coupon notes or bonds, with a defined payment cycle of semi-annual payments to the holder.

U.S. Treasuries have characteristic properties that require special attention for the purposes of the present invention and therefore are used in the following discussions. One important attribute of treasuries, in the context of the present invention, is the minimal and uniform default risk; the issuance of U.S. government paper removes the default risk as a defining criteria in the relative pricing of treasuries in the market place. However, other fixed income securities have varying levels of default risk, and the amount of risk determines the value and acceptability of such a security to a particular financial portfolio. For example, some investment managers may have restrictions on the amount and character of risk for a particular account. Further, securities issued with foreign currencies have a further currency risk apart from the interest rate and default risks.

Treasuries are auctioned by the U.S. government at pre-established auction dates. The price for the treasuries having a face value with a set coupon rate will define the actual yield of the security. After the auction, the treasuries enter the secondary market and are traded typically "over the

counter", i.e., without a defined exchange. As inflation expectations and market conditions change, the prices of the recently auctioned treasuries fluctuate. These price changes are reflected by competing bid and ask prices communicated among brokers and dealers in the secondary market. For example, the yield of a given treasury increases as its price drops in the market reflecting an overall increase in the interest rates for that term of security. With other fixed income securities, the credit worthiness and rating of the issuers may also have an effect on the market price for the security.

The newly auctioned securities are traded with and in conjunction with the securities issued in earlier auctions. In this context, some securities are traded more often than others and are called the "actives"; these usually correspond to the recent issues as opposed to the older securities in the market. Indeed, some older securities are infrequently traded, creating an illiquid market that may or may not reflect the true market determined interest rate for that maturity length security. When private organizations issue bonds, typically through underwriters, the bond is initially priced based on the interest rate, the default risk, and possibly other factors relating to the legal rights relating to the bonds (which may be subordinated to other debt, or may be convertible into stock). Corporations are rated for their credit worthiness by private companies, and the pricing of their securities can rise or fall with changes in that credit rating.

Treasuries are sold by the government to fund projects, mandated payments and make strategic investments that cannot be paid by current receipts. Corporate bonds are sold to raise capital for the operations of the corporation, although some private bond offerings are for specific assets. Treasuries and corporate bonds are purchased by individuals and institutions for a variety of reasons, including the protection of principal with a low risk investment vehicle and the generation of known future cash flows to fund the

needs of a specific group of investors, e.g., pension participants. Many of these accounts have detailed limits and restrictions on the amount of risk in the account portfolio, and may be subject to government regulation on the types and amounts of securities they may include.

As can be realized by the foregoing description, the very size and diversity of the securities market implicates an unprecedented level of sophistication by market participants in the pricing and transactions involving these securities. The very complexity associated with the transactions and the scale of trading undertaken by institutional participants necessitates a rigidly structured approach in trading. The capital at stake and the fluidity of future commitments makes it critical to have a method of measuring the performance of portfolio managers, so that plan sponsors for the pension plans and the like can precisely determine whether the capital under their control is properly invested.

In the past, the only barometer for fixed income investing was the stated price and yield for one or more specific instruments such as the 30 year treasury bond. These yield values would be quoted on an ad hoc basis as a general measure of market position and direction. More recently, several large brokerage houses have developed different indices to track the fixed income market beyond the single price issue. For example, Shearson-Lehman American Express has developed a T-Bond index value that calculates a weighted average of every bond in circulation. Other indices exist with similar mechanisms for tracking the credit marketplace.

There are several significant drawbacks to the use of these forms of indices. The actual value is calculated at the close of the financial markets and, therefore, is not a real time determination, and, in fact, rapidly becomes

stale as trading continues overseas and during the next trading day in the United States.

Other problems also exist; taking the entire market into account necessarily includes lightly traded issues that skew the final value from extant market conditions. This is so as these lightly traded issues do not accurately reflect the term structure of interest rates as other investment criteria, e.g., tax implications, control their market price.

There has also been a significant need for a hedging instrument on fixed income investing. In this context, an investor might purchase a portfolio of long term bonds that are sensitive to small changes in interest rates; to hedge this investment, this investor would enter a futures contract to sell instruments at a specific date in the future. Alternatively and more desirably, the hedge could be made with an index corresponding to a defined set of securities. This is not practical with the presently available indices due to their reliance on a broad spectrum of securities in the defining basket; this precludes effective utilization of these indices as a basis for trading futures or option contracts.

From the above, it is apparent that there remains a substantial void in the credit markets and a corresponding need for a real time barometer of the fixed income securities marketplace for the evaluation of portfolio performance, the trends and current market conditions, and the trading of indexed future and option contracts for fixed income securities.

As can be realized by the foregoing description, the very size and diversity of the treasury market implicates an unprecedented level of sophistication by market

participants in the bidding, offering, buying, and selling transactions involving these securities. The very complexity associated with the transactions and the scale of

trading undertaken by banks, brokers, dealers and institutional participants necessitates a rigidly structured approach to trading.

In the past, open outcry auction bond brokering has served its customers well, providing highly efficient executions at near perfect market pricing. The open outcry auction applied to bond trading was implemented by a broker working with a collection of customers to create and manage a market. Typical customer representatives--both buyers and sellers--at a common location (e.g., a single room) where the representatives of the customers would communicate with each other to develop pricing and confirm transactions. This process employed the expression by the representatives of various bid and offer prices for the fixed income security at select volumes (i.e., how many million dollars of bonds at a given maturity). This expression would involve the loud oral "cry" of a customer-proposed bid or offer and the coordination with the fellow representatives regarding the extraction of complimentary positions--until a transaction match is made and a deal is done. This "trade capture" process relies on after-the-fact reporting of what just transpired through the oral outcry trade.

Recently, the trade capture process was performed by having designated clerks input data into electronic input devices. An input clerk would attempt to interpret the open outcry of many individual brokers simultaneously who sequentially are making verbally known their trading instructions of their customers. The quality of the data capture was a function of the interpretative skill of the input clerk, and the volume and the volatility of customer orders. A significant drawback to this type of auction data capture process is the difficulty in discerning the distinct trading instructions verbalized in rapid succession during a quickly moving market, so that an accurate sequence of data can be captured by brokers and a set of inputters.

Problems exist in open outcry auction that deplete efficient trading. The speed at which trading flows and the oral nature of the auction process injects a

potential for human error that often translates into many millions of dollars committed to trades unrelated to customer objectives. As such, the broker is left at the end of each trading day with a reconciliation process that may, under certain market conditions, wipe out all associated profit from that day's trading. Also, customers may quickly change direction regarding trading, based on new information available to the market. Shifting position or backing out of previously committed transactions on very short notice is often very difficult in the traditional open outcry auction process.

There are many conventional efforts to incorporate computers into trading support for select applications and securities. Almost all trading today involves some computer support, from simple information delivery to sophisticated trading systems that automate transactions at select criteria. However, these systems have not significantly impacted the issues presented above as they relate to open outcry auction trading in the fixed income field. Conventionally, applications have been designed for discrete segments of the overall securities market because of the high level of business-specific knowledge associated with each. Also contributing to this approach has been the absence of a commercial software infrastructure or middleware layer that could support a more fully integrated solution. It was with this understanding of the problems with certain trading processes that formed the impetus for the present invention.

SUMMARY OF THE INVENTION

The present invention is a trading system and method which allows for real time processing of transaction information from a variety of sources and destinations. The invention uses an object oriented architecture to allow for imbedding numerous components into the trading system without requiring modification to existing applications. The persistence control of the invention handles real time events using a messaging system and object managers which coordinates objects instances and distribute and update object environments as appropriate. With the present invention, objects may be located on distributed systems throughout a network, and objects do not need location information to interact with distributed objects.

The object oriented system of the present invention is organized into "objects", each comprising a block of computer instructions describing various procedures ("methods") to be performed in response to "messages" sent to the object or "Events" which occur with the object or the computer system. ActiveX is the name Microsoft has given to a set of strategic object oriented program technologies and tools for its Windows 95/98 and Windows NT operating system. JAVA is the name Sun Microsystems has given to its definition of an object oriented virtual machine and corresponding programming language.

The Component Object Model (COM) may be used in a network. Object Linking and Embedding (OLE) is Microsoft's program technology for supporting compound documents such as the Windows desktop. ActiveX controls are among the many types of components that use COM technologies to provide interoperability with other types of COM components and services. ActiveX controls are the third version of OLE controls (also called OCX), providing a number of enhancements specifically designed to facilitate distribution of components over high-latency networks and to provide

integration of controls into Web browsers. Microsoft now uses the term "ActiveX control" instead of "OCX" for the component object. One of the main advantages of a component is that it can be re-used by many applications (referred to as "Component Containers"). The present invention provides additional advantages by providing a persistence control for the component of the system. A COM component object (ActiveX control) can be created using one of several languages or development tools, including Visual C++ and Visual Basic, or PowerBuilder, or with scripting tools such as VBScript. terminologies and still managed with the persistence control of the present invention.

The present invention, in one embodiment, utilizes object oriented system and the ActiveX control technology combing with sophisticated calculation algorithms to provide a middleware level interface between end user applications and external information sources, such as distributed data bases, digital data feeds, and electronic communications networks (or "ECNs") used for financial trading. More particularly, the software components of the present invention allow interfaces with commercially available data feeds, internal and external networks, and legacy systems used by the international financial community. The ActiveX persistence control makes it easy for financial vendors, traders, investment managers, insurers, researchers and others to include the complex real-time prices, P&L, risk factors, hedge ratios, duration, credit contributions, and many other analytics in their own applications.

The present invention, in one form, relates to a security management and trading system for fixed interest securities. The system operates in conjunction with a data processing system for transacting the purchase and sale of at least one security having a set of characteristic data. The data processing system is operated by a plurality of trading participants and

transmits information relating to security characteristic data. The trading system comprises a network, a workstation, and a server. The workstation is coupled to the network and includes a display for presenting to a participant information about pending market conditions as they relate to the security characteristic data of a security. The server is in communication with the workstation over the network and provides event information. At least one of the workstation and server include an object oriented software program comprising a plurality of objects, with each object including a plurality of instructions and associated data. The workstation includes at least one securities evaluation software application which references an object having time sensitive security characteristic data. The workstation also includes persistent control software having instructions for enabling the workstation to update the object having the time sensitive security characteristic data when external event information is received and the external event information is relevant to the time sensitive securities data.

The present invention, in another form, is a method for of managing and trading fixed interest securities over a computer network. This includes transacting the purchase and sale of at least one security having a set of characteristic data. The data processing system is operated by a plurality of trading participants and transmits information relating to the securities characteristic data. At least one trading participant has a workstation with a securities evaluation software application. The method includes providing a securities evaluation software application which references an object having time sensitive security characteristic data; providing persistent control software for updating the object having the time sensitive security characteristic data; preventing access to the securities evaluation software application except through the persistent control software; and receiving

external event information and the persistent control determining whether the external event information is relevant to said time sensitive securities data.

Further aspects of the present invention involve the workstation further including object manager software. The object manager software may include update software for modifying data in an object in response to external event information. The persistent control may include a database referencing a plurality of objects. That database may include state information relating to the plurality of objects. The server may include a messaging system which allows at least one of the plurality of objects to be located on the server and be referenced by the securities evaluation software application. The server may be configured to interact with a plurality of ECNs. The workstation may include virtual market software for coordinating said plurality of ECNs for said securities evaluation software application.

Another aspect of the invention relates to a machine-readable program storage device for storing encoded instructions for a method of according to the foregoing method.

BRIEF DESCRIPTION OF THE DRAWINGS

The above mentioned and other features and objects of this invention, and the manner of attaining them, will become more apparent and the invention itself will be better understood by reference to the following description of an embodiment of the invention taken in conjunction with the accompanying drawings, wherein:

Figure 1 is a schematic diagrammatic view of a Server/Workstation system using the present invention.

Figure 2 is a detailed block diagram of the operation of the present invention relating to the logical model of the software programming.

Figure 3A is a schematic diagrammatic view of the prior art system relating to electronic securities trading.

Figure 3B is a schematic diagrammatic view of the system of the present invention relating to securities trading.

Corresponding reference characters indicate corresponding parts throughout the several views. Although the drawings represent embodiments of the present invention, the drawings are not necessarily to scale and certain features may be exaggerated in order to better illustrate and explain the present invention. The exemplification set out herein illustrates an embodiment of the invention, in one form, and such exemplifications are not to be construed as limiting the scope of the invention in any manner.

DESCRIPTION OF THE PRESENT INVENTION

The embodiment disclosed below is not intended to be exhaustive or limit the invention to the precise form disclosed in the following detailed description. Although specific versions of commercial software packages are noted in the following description, the invention may be embodied in conjunction with other versions or other software packages that perform the required functions of the inventive software. Rather, the embodiment is chosen and described so that others skilled in the art may utilize its teachings.

The detailed descriptions which follow are presented in part in terms of algorithms and symbolic representations of operations on data bits within a computer memory representing alphanumeric characters or other information. These descriptions and representations are the means used by those skilled in the art of data processing arts to most effectively convey the substance of their work to others skilled in the art.

An algorithm is here, and generally, conceived to be a self-consistent sequence of steps leading to a desired result. These steps are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise

manipulated. It proves convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, symbols, characters, display data, terms, numbers, or the like. It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely used here as convenient labels applied to these quantities.

Some algorithms may use data structures for both inputting information and producing the desired result. Data structures greatly facilitate data management by data processing systems, and are not accessible except through sophisticated software systems. Data structures are not the information content of a memory, rather they represent specific electronic structural elements which impart a physical organization on the information stored in memory. More than mere abstraction, the data structures are specific electrical or magnetic structural elements in memory which simultaneously represent complex data accurately and provide increased efficiency in computer operation.

Further, the manipulations performed are often referred to in terms, such as comparing or adding, commonly associated with mental operations performed by a human operator. No such capability of a human operator is necessary, or desirable in most cases, in any of the operations described herein which form part of the present invention; the operations are machine operations. Useful machines for performing the operations of the present invention include general purpose digital computers or other similar devices. In all cases the distinction between the method operations in operating a computer and the method of computation itself should be recognized. The present invention relates to a method and apparatus for operating a computer in processing electrical or other (e.g., mechanical, chemical) physical signals to generate other desired physical signals.

The present invention also relates to an apparatus for performing these operations. This apparatus may be specifically constructed for the required purposes or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The algorithms presented herein are not inherently related to any particular computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description below.

The present invention deals with "object-oriented" software. The "object-oriented" software is organized into "objects", each comprising a block of computer instructions describing various procedures ("methods") to be performed in response to "messages" sent to the object or "events" which occur with the object. Such operations include, for example, the manipulation of variables, the activation of an object by an external event, and the transmission of one or more messages to other objects.

Messages are sent and received between objects having certain functions and knowledge to carry out processes. Messages are generated in response to user instructions, for example, by a user activating an icon with a "mouse" pointer generating an event. Also, messages may be generated by an object in response to the receipt of a message. Further, external stimuli, such as an ECN trade, may cause an object to generate an event. When one of the objects receives a message, the object carries out an operation (a message procedure) corresponding to the message and, if necessary, returns a result of the operation. Each object has a region where internal states (instance variables) of the object itself are stored and where the other objects

are not allowed to access. One feature of the object-oriented system is inheritance. For example, an object for drawing a "circle" on a display may inherit functions and knowledge from another object for drawing a "shape" on a display.

A programmer "programs" in an object-oriented programming language by writing individual blocks of code each of which creates an object by defining its methods. A collection of such objects adapted to communicate with one another by means of messages comprises an object-oriented program. Object-oriented computer programming facilitates the modeling of interactive systems in that each component of the system can be modeled with an object, the behavior of each component being simulated by the methods of its corresponding object, and the interactions between components being simulated by messages transmitted between objects.

An operator may stimulate a collection of interrelated objects comprising an object-oriented program by sending a message to one of the objects. The receipt of the message may cause the object to respond by carrying out predetermined functions which may include sending additional messages to one or more other objects. The other objects may in turn carry out additional functions in response to the messages they receive, including sending still more messages. In this manner, sequences of message and response may continue indefinitely or may come to an end when all messages have been responded to and no new messages are being sent. When modeling systems utilizing an object-oriented language, a programmer need only think in terms of how each component of a modeled system responds to a stimulus and not in terms of the sequence of operations to be performed in response to some stimulus. Such sequence of operations naturally flows out of the interactions between the objects in response to the stimulus and need not be preordained by the programmer.

Although object-oriented programming makes simulation of systems of interrelated components more intuitive, the operation of an object-oriented program is often difficult to understand because the sequence of operations carried out by an object-oriented program is usually not immediately apparent from a software listing as in the case for sequentially organized programs. Nor is it easy to determine how an object-oriented program works through observation of the readily apparent manifestations of its operation. Most of the operations carried out by a computer in response to a program are "invisible" to an observer since only a relatively few steps in a program typically produce an observable computer output.

In the following description, several terms which are used frequently have specialized meanings in the present context. The term "object" relates to a set of computer instructions and associated data which can be activated directly or indirectly by the user. The terms "windowing environment", "running in windows", and "object oriented operating system" are used to denote a computer user interface in which information is manipulated and displayed on a video display such as within bounded regions on a raster scanned video display. The terms "network", "local area network", "LAN", "wide area network", or "WAN" mean two or more computers which are connected in such a manner that messages may be transmitted between the computers. In such computer networks, typically one or more computers operate as a "server", a computer with large storage devices such as hard disk drives and communication hardware to operate peripheral devices such as printers or modems. Other computers, termed "workstations", provide a user interface so that users of computer networks can access the network resources, such as shared data files, common peripheral devices, and

inter-workstation communication. Users activate computer programs or network resources to create "processes" which include both the general operation of the computer program along with specific operating characteristics determined by input variables and its environment.

Because of the time sensitive nature of information used in conjunction with the system of the present invention, the workstations having the object oriented software program of the present invention specifically have software that maintains the persistence of objects in the workstation's object space. As used herein, the term "Persistence" relates to an instance of an object which has been created in a particular environment and has specific securities related data which is time sensitive. As the object dynamically interacts with its temporal environment, the object's associated data changes. As new information becomes available in other parts of the computer network, events are generated which could be relevant to the object's associated data. However, that associated data should not be automatically updated because the generation of events in the computer network may not directly correspond to the real-time occurrence of changes to the external characteristics represented by that data. For the present application, the term "persistence" relates to the maintenance of an object's correct temporal state by only updating an object's associated data when the data update corresponds to the temporal state of that object. Persistence implies some kind of temporal permanence, whether or not that permanence is implemented via server/workstation hard drives, server/workstation memory, or some other kind of information storage/retrieval mechanism, including but not limited to electronic or manual communications. The persistence semantics is

transparent to both the persistent control and applications, and is affected via object managers.

The terms "desktop", "personal desktop facility", and "PDF" mean a specific user interface which presents a menu or display of objects with associated settings for the user associated with the desktop, personal desktop facility, or PDF. When the PDF accesses a network resource, which typically requires an application program to execute on the remote server, the PDF calls an Application Program Interface, or "API", to allow the user to provide commands to the network resource and observe any output. The term "Browser" refers to a program which is not necessarily apparent to the user, but which is responsible for transmitting messages between the PDF and the network server and for displaying and interacting with the network user. Examples of Browsers compatible with the present invention include the Navigator program sold by Netscape Corporation and the Internet Explorer sold by Microsoft Corporation (Navigator and Internet Explorer are trademarks of their respective owners). Although the following description details such operations in terms of a graphic user interface of a Browser, the present invention may be practiced with text based interfaces, or even with voice or visually activated interfaces, that have many of the functions of a graphic based Browser.

The present invention also relates to an apparatus for performing these operations. This apparatus may be specifically constructed for the required purposes or it may comprise a general purpose computer as selectively activated or reconfigured by a computer program stored in the computer. The algorithms presented herein are not inherently related to any particular

computer or other apparatus. In particular, various general purpose machines may be used with programs written in accordance with the teachings herein, or it may prove more convenient to construct more specialized apparatus to perform the required method steps. The required structure for a variety of these machines will appear from the description below. In particular, although the disclosed embodiment deals with Microsoft's ActiveX environment, the present invention is equal, applicable to other object oriented operating systems such as JAVA compatible or LINUX based systems. Similarly, although the present invention is described with a specific embodiment referencing Microsoft's Windows NT operating system, the invention is compatible with Microsoft's Windows 2000 operating system and is envisioned to be compatible with future operating systems.

The inventors of the present invention have previously created systems which provide a facility for controlling the persistence of objects in a static environment. Such systems have operated in an environment that was not subject to significant time constraints. However, when trading securities in real-time in the world-wide securities markets, the time relationship of data can be of critical importance and the immediacy of updating such data becomes significant. Thus, in order to deliver immediate updating and response to the desktop of the trader, the inventors needed to further develop such persistence controls to operate in the real-time trading environment.

A general schematic view of the present invention is illustrated in Figure 1. External Information 10 represents the various electronic resources that may be available to Server System 12, such as the Internet, ECNs, enterprise data bases available over a WAN, electronic news or information services, and other possible sources of information. Server System 12 is coupled to Workstation 14, and in actual implementation several hundred workstations 14 may be coupled to a single server system 12. External

Information 10, Server System 12, and Workstation 14 are configured and structured to support the work of a participant in Securities Markets 16, representing the variety of fixed income securities markets available world wide. Ultimately, a participant accesses the information and executes trades or other instructions electronically through user workstation display 18.

In accordance with the present invention, workstation 14 includes a persistence control and an object manager that receives and acts on event information received from server system 12 and/or external information 10. Every application on the desktop of user workstation display 18 must access objects through the persistence control. When event information is received by the persistence control, underlying data of the objects on workstation 14 which do not have the updated data are so updated by the object manager. When event information is received by the persistence control but the information is old in relation to the data already in the object, the persistence control discards this information. Also, an event received by the persistence control may have implications for several objects on workstation 14, causing the persistence control to activate the object manager to update each object separately.

The persistent control is utilized by applications to perform the locating, querying, creating, modifying and deleting of objects. It is also responsible for distributing update events for those objects, in which the application is interested, to the application. The persistent control provides the focal point of management of objects for an application. The objects themselves utilize the persistent control to affect their creation, modification and deletion, although the role of the persistent control is transparent to the application. The objects managed by the persistent control are COM objects, and are directly manipulated by applications, although such manipulation is not made persistent until the application requests the object's state to be so. A

persistent control manages its 'space' of objects separate to the 'space' of objects of other persistent controls, even those within the same application thread. An application may utilize any number of persistent controls to create separated simultaneous states of computation.

The object managers are responsible for locating, querying, creating, modifying and deleting objects on behalf of persistent controls. They are also responsible for managing update events to those objects and informing those persistent controls that are interested in such object events. An update event is currently defined to include, but is not limited too, object creation, modification and deletion. The object manager enforces the required persistence semantics for object classes. The mechanism by which the object manager implements the required persistence semantics for the object classes it manages is transparent to the persistent control, and thus the applications. A single class of objects may be managed by numerous object managers over time without impacting either the persistent control or underlying applications, although a class of objects can be assigned to only one object manager at any one time.

The persistent control acts as an interface between the applications and the objects available on the workstation and around the network. The applications may not necessarily know where objects are located, either on the workstation, on the server, or externally but in communication with the server and/or workstation. When the application requests an operation on an object, the persistent control activates the object manager to perform the operation, if appropriate. If an application desires the creation of a non-persistent object, the persistence control has the object manager create an object which is delivered to the application as a COM object which the application may manipulate without restriction. However, once an application requests that an object have persistence, that application may only

manipulate the object through the persistent control. The persistent control maintains a database of the objects with their associated class, object identification, object manager, and state information. With this information, the persistent control can track the state of each object, and perform any required manipulations to the objects by invoking the object manager.

The present invention features a three-tier, distributed architecture that effectively decouples the "front-end" user interface, "mid-level" business rules, and "back-end" database systems. The result provides a real-time, global, multi-currency, multi-product system that is economical to operate, extend, and support. The system of the present invention employs a number of commercial, "off-the-shelf" products in its core architecture that are considered to be either actual or de facto industry standards. These technologies include:

Message-Oriented Middleware - The system makes extensive use of transactions processing software, which in the exemplary embodiment is the *Tuxedo Transaction Processing (TP) Monitor by BEA Systems,* Inc. This Message-Oriented Middleware (MOM) provides reliability and scalability when processing transactions. In addition, it provides the ability to easily add and remove transaction processing services.

Reliable Messaging - The system uses a real time data service to obtain real-time securities information relating to the securities being managed and traded by the participant at user workstation display 18. The exemplary embodiment of the invention utilitizes the *Reuters SSL 4.* 0 to deliver such information and achieve reliable messaging. This technology provides scaleable message services that enable workstation applications to subscribe to data events.

Relational Database Services - The system uses conventional relational database techniques to store information needed at the level of

System Server 12. The exemplary embodiment of the invention utilizes the *Oracle 8.0.4 Relational Database Management System (RDBMS).* This provides the ability to maintain persistent transactions in a reliable, scaleable manner at the server level.

Workstation Operating System - The exemplary embodiment of the present invention was developed for the *Microsoft NT 4. 0* operating system. This environment provides the application environment required by modem trading systems.

Object Component System - The exemplary embodiment of the invention employs the Microsoft *Component Object Model* (COM). This model enables the product to support rapid application development and deliver high performance execution on the workstation.

The underlying design theme of the present invention is componentization, and this applies to both its server and workstation levels. For example:

- Server business processes are components *(Tuxedo* services). As such, they can be invoked or terminated as necessary. Moreover, they can be reconfigured, added or removed on the fly. This flexibility means that Server System 12 can maintain 7x24 operations, while still being able to meet the demands of rapidly changing markets.
- Workstation objects are components (COM objects). Objects can be added, removed or modified at the workstation level without the need to reconfigure applications. The system of the present invention has object management software to expose vital details to applications that can interrogate them at runtime. The ability to perform runtime discovery allows workstations to expand their functionality (e.g., instrument coverage) without the need to modify underlying code. This

is possible because the system's financial applications require only a minimal understanding of object semantics.

Message-Oriented Middleware and Component Objects

The present invention's ability to provide the high degree of scalability and resilience needed by modern trading systems while simultaneously supporting extensible, performing applications required the inventors to meld the best attributes of the system's underlying technologies. This was accomplished by combining the time-tested scalability and resilience of the *Tuxedo* MOM with the demonstrated scalability of *Reuters* SSL with the framework of the MS COM on the workstation.

The extensibility and performance of the COM in streamlined call interfaces to well defined objects plus the system's own COM extensions permit the product's infrastructure to transport objects via *Tuxedo* messaging. Since neither this transport mechanism nor Tuxedo's messaging services are coupled with the semantics or contents of the system's objects, the system can be highly extensible while leveraging off each technology's fundamental strength.

The Principle of Locality

The principle of locality was exploited when designing the present invention in order to provide it with high processing performance in a distributed environment. There are four types of locality: (1) Spatial Locality - the closeness in space, or proximity; (2) Temporal Locality - closeness in time; (3) Degree Locality - closeness in capacity; (4) Effectual Locality - closeness in purpose.

The locality principle holds that optimal system performance is achieved when servers, workstations, and other resources (executables/libraries, databases, processors, memory, networks, and

storage devices) are designed or configured to closely match the demands of applications.

The present invention leverages the principle of locality by situating the most mission-critical business data and system components where they are needed most - on the workstations of traders who require immediate decision support services. Conversely, data most critical for executing business transactions is located on the server. Specifically, the product exploits the locality principle in the following ways:

- By expeditiously transferring data required by traders to their workstations, where system components can operate on it directly;
- By prepositioning the most time-critical data on workstations to minimize retrieval and presentation delays;
- By situating all of the data needed by individual traders during the normal course Of business as close to their workstations as possible. This significantly reduces the time required to conduct normal operations;
- By placing on workstations only data of direct interest to traders and leaving on the server that which is utilized for business transactions.

The above techniques are used to make the present invention the most performing and responsive trading system available without sacrificing reliability, scalability, and extensibility.

A more detailed view of the software components of Figure 1 is illustrated in the logical model representation of Figure 2 of the product embodying the present invention: RAPTr-NT. The components of Server System 12 are generally shown interacting with External Information 10

through the WAN, and interacting with Workstation 14 through the LAN. Workstation 14 uses Trading System Applets to provide information on User Workstation Display 18.

Server System 12 includes storage in the form of distribution storage (from routers and distribution servers) and a relational database management system ("RDBMS") used by the Tuxedo messaging service to store and index message information. The Tuxedo service, or any other suitable computer messaging system, both relays messages to their destinations and tracks information about those messages. In effect, the messaging system obtains and sorts messages and tracks where those messages are supposed to be delivered.

Workstation 14 includes three layers of software, with the application layer including all the applications for the market participant, including the Analysis, Blotter, Risk, Account, Summary, Deal Capture, Calculator, and Explorer functions. These functions may be presented to the market participant through the Trading System Applets displayed on user workstation display 18. The invention involves the combination of the Persistent Control, through which all applications execute, and the object manager interfaces. Messages received from Server System 12 do not directly effect the operation of the application workspace, rather the Persistent Control portions of the workstation software determines how and when such messages change the data of any of the objects in the application workspace. For example, the Persistent Control may receive a message which causes it to perform one or more of the following functions: find an existing object; create a persistent object; update an object state; query for the existence of an object;, instantiate an object; or make an instantiation a persistent object. Further, the object manager software may also have predefined methods that it may operate on data received from Server System 12. The Persistent Control

may run in an event loop, whereby events are processed by class, then by rule or other methodology, until all events have been dealt with.

The advantages of the present invention can be shown by the differences between the logical arrangement of Figure 3A, illustrating the conventional arrangement of traders and portfolio managers, with the logical arrangement of the present invention shown in Figure 3B. In Figure 3A, traders and portfolio managers perform the buying and selling of securities over several ECNs, requiring several interconnections of the ECNs and various market participants. Once a trade is consummated on one of the ECNs, the back office mainframe computers exchange the relevant securities and money over conventional electronic clearing agency or clearinghouse such as Mortgage Backed Securities Clearing Corporation ("MBSCC"), Government Securities Clearing Corporation ("GSCC"), or the Depository Trust Company ("DTC"). The difficulty for the trader or portfolio manager involves the fact that trades on one ECN may effect the valuations for trades that are being contemplated on one of the other ECNs.

This difficulty is addressed by the RAPTr-NT product by utilizing the present invention to update the trader or portfolio manager's display to reflect recent trades on the other ECNs. This is shown in Figure 3B by virtual market 20, wherein the combined effects of each ECN is reflected in real time on the user workstation display of the trader or portfolio manager. While the back office operation of mainframe computers and clearinghouses operates in the conventional manner, the present invention provides the trader or portfolio manager with information as current as possible so that contemplated trades can be evaluated in light of the most recent information from sources outside of the ECN on which the trade is being contemplated.

RAPTr-NT is designed with a layered approach which permits the product's various technologies to be effectively decoupled from one another.

This decoupling yields significant benefits in terms of architectural flexibility. For example, RAPTr-NT's Oracle RDBMS can be slotted out and replaced with Informix without affecting client performance and only minimally affecting the server. This layering concept applies to both the product's Physical Model (hardware and software processes) and its Logical Model (API's and software interactions).

RAPTr-NT's server processes support the persistent storage of information and the distribution of data and updates to both the Local Area Network (LAN) and the Wide Area Network (WAN). There are five distinct types of processes that run on servers:

1. *Tuxedo* Services - These are standard Tuxedo processes that perform database queries and updates at the behest of a client. Note that such a can client could be a process running on either a server process or a workstation process. As a result of these services being managed by Tuxedo, they receive all the benefits of the Tuxedo TP Monitor, such as:
    a. Startup/shutdown on demand;
    b. Restart on failure;
    c. Load balancing across multiple servers.

RAPTr-NT's *Tuxedo* services use the Oracle Relational Database Management System (RDBMS) to perform database work. One *Tuxedo* service may interact with other *Tuxedo* services to process a transaction. A *Tuxedo* service has two responsibilities beyond those associated with the execution of transactions:

d. Invocation of the Global Distribution Service, which is responsible for distributing transactions on a global basis;

e. Invocation of the Publish/Subscribe Service, which causes the distribution of object state changes to all interested subscribers. This means that all RAPTr-NT clients are continually refreshed with the latest available object states.

2. Global Replication and Routing Services - These services provide the replication of transactions and then distribute them to the WAN in a guaranteed manner. The result is that transactions applied to the local database are guaranteed to be applied to other databases that wish to receive them. Such transactions include trades, firm prices, instrument updates, and account information. (The Global Replication and Routing Services are not performed when RAPTr-NT is installed in a single-site configuration.)

3. Publish/Subscribe (Reliable) - This process is responsible for publishing non-critical data updates to subscribers on the LAN (e.g., market updates or other similar types of data). Publish and subscription services are carried out by *Reuters* SSL, which provides a scaleable solution that can meet the needs of very large installations. (Note that a data subscriber can be a server process or a workstation process.) Data published via reliable semantics will be delivered to clients whenever possible. However, there is some a possibility that messages will become lost from time to time due to network load or publisher failure. In other words, the delivery of *Reliable* data is not guaranteed.

Typical examples include price and analytic updates that can usually be refreshed by client processes as soon as the next tic becomes available or when the client is notified of the failure and re-requests the data;

4. Publish/Subscribe (Guaranteed) - This process is responsible for publishing critical data updates to subscribers on the LAN. The publishing of and subscription to such data are accomplished using the *Tuxedo TP* Monitor. Data published using guaranteed semantics will always be delivered to the client. Examples include trade, position, and risk data. Guaranteed messages are queued for individual clients until they are ready to receive them. As was the case with reliable data, a subscriber can be either a workstation or a server process;

External Interfaces - These processes communicate with the *Tuxedo* services in exactly the same manner as workstation processes, but they do so while running on a server. Typically, external interface processes are used to effect the two-way transfer of trade data between RAPTr-NT and a firm's back-office system. External interfaces can also be used to bridge between RAPTr-NT and outside data sources or sinks.

RAPTr-NT workstation processes support user applications. They also abstract applications from the underlying transaction/data delivery model. There are four distinct types of workstation processes:

1. Messaging Interface - The Messaging Interface process is responsible for submitting transactions and queries to *Tuxedo* and for receiving subscription updates from *Reuters* SSL. It acts as a standard data access mechanism for the RAPTr-NT workstation, which effectively isolates it from all data sources.

Because of this, the TP Monitor and/or the subscription update infrastructure can be swapped out without impacting the application software;

2. Object Managers - Object Managers are standard processes by which applications initiate transactions and queries on data objects. The Object Managers are responsible for imposing specific transaction semantics on particular groups of objects. Currently, there are two types of Object Managers in RAPTr-NT:

   a. Local Object Manager - The Local Object Manager is responsible for handling transactions made to objects stored in a workstation's memory and publishing the data to all interested application processes. Since local objects are kept in memory, they are not stored between boot sessions. They are therefore used as "scratch" objects or to facilitate communications between applications.

   b. Transacted Object Manager - The Transaction Object Manager is responsible for handling transactions to objects accessed through the Message Interface (i.e., *Tuxedo)* and disseminating updates received from the Message Interface to interested applications on the workstation. Since these objects are transacted using Tuxedo, they are fully ACID-comformant (Atomic, Consistent Isolated, and Durable).

3. In addition to the above Object Managers, it is possible to add others to RAPTr NT as the need arises. These could include specialized functions. that can access local databases/files or perhaps legacy systems through specialized protocols;

4. Trading System - The Trading System process is RAPTr-NT's principal user application. Multiple Trading System processes can run concurrently without negatively impacting data consistency. In fact, two Trading System processes behave in the same manner regardless of whether they are running on the same workstation or on multiple workstations on the LAN;

5. Miscellaneous Applications - In addition to the principal processes described above, RAPTr-NT can support a wide variety of Productivity Applications. These can include commercial products such as MS Word, Excel, and Access, as well as many other vendor products and virtually all manners of compliant homegrown applications.

6. It is important to note that since the workstation infrastructure is built using standard COM, RAPTr-NT is accessible from:

   a. Applications built with Visual Basic for Applications (VBA), a development language that is included with all the Microsoft productivity applications such as Word, Excel, and Access. This means that Excel Spreadsheets can access RAPTr-NT to facilitate proprietary transactions and receive updates in exactly the same manner as the product's own applications.

   b. In-house developed applications written in C++, Visual Basic, Delphi or COM-connected Java;

   c. Other applications that support VBA or COM Automation objects.

RAPTr-NT's Logical Model utilizes a layered architecture that enables its various hardware and software components to be decoupled from one another. This decoupling greatly facilitates extending and modifying the

system and changing its underlying technologies (e.g., substituting Informix for Oracle, etc.). It also enhances the productivity of workstations by enabling users to run a wide variety of commercial products such as MS Word and Excel in conjunction with all of RAPTr-NT's trading applications.

Workstation processes can be generally classified into the following groups:

1. Server Software and APIs - This segment implements the server services and processes that handle RAPTr-NT's business functions;
2. Workstation Software and APIs - This segment constitutes RAPTr-NT's workstation infrastructure;
3. Trading Applications - This segment contains the software and API's that implement RAPTr-NT's main end user functions.

RAPTr-NT's server supports the execution of the product's business rules, manages its database, and distributes its updates. The following processes perform these functions:

1. Publish/Subscribe (Reliable) Process - This process is responsible for distributing object data updates to the LAN. It does so by waiting for Tuxedo messages that have been converted to *Reuters* SSL updates. These messages are delivered in a reliable manner via the SSL messaging system. If an error occurs within the Publish/Subscribe process, queued messages will be lost. Clients are immediately informed of such failures, whereupon they may refresh their data objects by re-requesting them (through *Tuxedo)*. In the event that the SSL itself detects a dropped message, it will inform clients of the failure so they can re-request the specified objects;
2. Publish/Subscribe (Guaranteed) Service - This *Tuxedo* service is responsible for distributing critical object data updates to the

LAN. It waits for appropriate messages, queues them as necessary, and distributes them to clients in a guaranteed manner. If an error occurs at the client queued messages are not lost and are transmitted from persistent storage when the client is able to receive them.

3. Global Distribution - This *Tuxedo* service receives transactions from the system's business processes and determines how each needs to be distributed around the world. If a transaction must sent to a remote domain, Tuxedo's global router would see that the process is accomplished properly.

In addition to the above services, there are two other types of services that perform RAPTr-NT-specific operations and manage firm interface requirements. These are as follows:

4. Tuxedo Services - These services perform all operations related to the management and processing of business data. While these are standard *Tuxedo* services, RAPTr-NT uses RogueWave DBTools++.h to abstract the database manipulation performed by *Oracle.*

5. External Interfaces - These are bespoke interfaces developed for RAPTr-NT customers that serve as data gateways, sources or sinks. An external interface can both submit transactions and receive updates via the Messaging Interface in the same manner as a workstation application. In fact, external interface processes operate peer-to-peer with all other external interfaces.

Workstations support the manipulation and display of data and the execution of RAPTR NT's business functions. In performing in this capacity, they make transaction calls to *Tuxedo,* and receive data notifications and

updates from *Reuters* SSL, although all applications are abstracted from *Tuxedo* and the SSL. There are a number of fixed processes that support the RAPTr-NT applications on the workstations, in addition to those that perform trading functions. The most important of these are as follows:

1. Messaging Interface Process - The Messaging Interface Process (MIP) is responsible for submitting transactions and queries to *Tuxedo,* and receiving subscription updates from the SSL. It also communicates with its clients (via either the Object Managers on the workstation or an external interface application on the server) to effect transactions and distribute updates.

The Message Interface exposes an API that allows clients to submit and receive data. The Message Interface is multithreaded which enables it to process incoming responses from *Tuxedo* and updates from the SSL at the same time that it is receiving responses from or distributing updates to it's client.

2. Object Manager Servers - The Object Manager Servers (OMS) present unified mechanisms by which applications initiate transactions and queries on data objects. OMS's are built as standard COM servers and use the COM to receive requests from and distribute updates to client applications. Currently, RAPTr-NT features three types of Object Managers:
   a. Local Object Manager - The Local Object Manager (LOM) is used to manage objects stored locally on the workstation. Objects managed by the LOM are regarded as "scratch objects" or communication channels that allow applications to interface with each other on the same workstation in a completely opaque manner. The LOM also supports object transaction semantics, which

ensures that all object modifications are performed in an atomic fashion

b. Transacted Object Manager - The Transacted Object Manager (TOM) is used to manage objects that have had transaction semantics enforced by a database and TP Monitor. These transactions conform to full ACID semantics and are completely controlled by the *Tuxedo TP* Monitor and the business services used to process the transaction.

The TOM also provides a caching/distribution mechanism for objects accessed by applications. It receives object updates from the Messaging Interface and then distributes them to the applications that requested diem. In the event that the update system fails, (i.e., SSL failure), it will send a message informing the applications of the situation and request that all affected objects be updated again.

c. Async Transacted Object Manager - The Async Transacted Object Manager (ATOM) is used to manage objects that have their transaction semantics enforced by a database and TP Monitor, but it differs from the TOM in that it causes transactions to be executed asynchronously with regard to the execution of the application.

The ATOM is used only for classes of objects where the strict transaction rules can be relaxed such as in the processing of rapidly changing market prices. These transactions still fully conform to ACID semantics, and they are completely controlled by the TP Monitor and applicable server business processes. However, the failure of such a transaction is deemed unimportant and therefore will not be communicated to applications.

The ATOM also receives object updates from the Messaging Interface and is responsible for disseminating them to interested applications. In the event that it is informed of a failure of the update system (i.e., SSL failure), it will send a message informing the applications of the situation and request that all affected objects be updated again.

It is important to note that the persistent semantics of objects that are managed by any Object Manager is exactly the same. Therefore, it is not possible for applications to know which Object Manager is controlling an object, unless the object wishes to expose that information. This theme is a fundamental building block of RAPTr-NT on the workstation.

The application software and APIs implement the Kestrel Trading System. It also exposes the RAPTr-NT objects to productivity applications and bespoke applications. All applications interact with the Workstation Software and APIs via a single ActiveX (COM) object. This ActiveX object is called the Persistent Control, and is the central interface for all applications. Note that all applications using a Persistent Control are *peers* within the system. All persistent objects within the RAPTr-NT are thread-safe, as are all support objects within the system.

A persistent object is one whose state can be modified by means of calls from the Persistent Control. When a persistent object is to be changed, a message is sent to the required Object Manager, which then uses its inherent transaction semantics (such as sending a transaction through the TP Monitor to a *Tuxedo* service) to modify the state of an object. Every persistent object has a fixed number of attributes that must be present. These are as follows:

1. **ProgID** - This is the COM object Programmatic Identifier string that identifies the type of the object. For example, **"KTC.1nstr.RegCpnX'** identifies a regular coupon instrument. The workstation infrastructure (specifically, the Persistent

Control) uses the ProgID to instantiate a copy of the COM object that contains its current state.

2. **ID** - This is the unique system identifier string by which the database recognizes an object. ID generation is completely up to the system.

3. **Version** - This is a serial number that corresponds to a unique object state. As the state of an object changes, its version number is incremented accordingly. When a transaction service receives a transaction request, it first inspects the target object's version number against the version number currently persistent in storage to ensure that the intended state change will be made to the correct object. If the version numbers fail to match, the transaction is rejected.

The Persistent Control plays a central role in all applications. It provides the mechanism by which applications locate, query, create, modify and delete persistent objects.

A single Persistent Control defines an *object space*. Objects within one Persistent Control's object space are completely separate and different from those in another Persistent Control's object space, even if both are running within the same process.

The result is that an application that can use a particular Persistent Control to manipulate objects can do so without impinging on the objects of another Persistent Control, *even if those objects represent the same underlying data object. This* goes for either Transaction Objects on the server and Local Objects on the workstation.

Effectively, each Persistent Control can manage objects that are in different states for a particular application. Applications using differing Persistent Controls do not need to lock objects to change their states, since

each is guaranteed unique. The result is that both applications and objects can be much simpler since they do not need to manage the locking process. All state changes are instead managed via the transaction semantics of the objects themselves. This is accomplished by either Tuxedo (if an object is managed by the TOM or the ATOM) or the local Manager.

The Persistent Control also exposes the following functionality to applications:

1. The ability to temporarily defer the transmission of object modifications from the Object Managers: When the deferral is over, all postponed modifications are batched to the appropriate object managers;
2. The ability to 'disconnect' objects from their Object Managers: This prevents modifications from being sent to the object managers and causes any updates that are received from such managers to be discarded. This allows objects managed by the Persistent Control to be used in "what if" scenarios. When the connection is reestablished, all objects are refreshed from their Object Managers' caches.
3. The ability to simulate external updates from within applications: This allows decoupled application processing to indirectly trigger business functions.

A Book Processor (also called the Workbench) is utilized by an open Book within the application to abstract the computations and processing required when the state of an object changes. A Workbench is a generic object that accepts work in the form of Workunits to process object events.

A Workunit in turn supplies the execution semantics required to process a unit of work. This may entail rearranging the sequence of operations required when multiple object events are received. The Workunit

uses an individual Workobject to effect the processing work of an individual object event. This usually requires the Workunit to reference a number other objects to complete its job.

For example, a Spread Workunit utilizes a number of Spread Workobjects, which provide the "object glue" that binds together two instruments and their quotes. The spread object itself contains the actual spread value (the difference between the prices of the two instruments). Changes to either instrument or the spread will cause the Spread Workunit to be activated, whereupon it will execute the Spread Workobject.

The Book Processors are extremely abstract processing engines. New Workunits can be added to Book Processors without impacting existing Workunits, and individual Workobjects can utilize and/or modify any other RAPTr-NT object as necessary. Modifications to objects by Workobjects are cascaded through the Persistent Control and the Book Processors.

The Trading System application is implemented using the Microsoft Foundation Classes (MFC) and a number of third-party MFC extension classes. The application can display a single page out of many, with each page consisting of any number of "Applet" views of a particular Book. The currently defined RAPTr-NT applet views include the following:

1. Analysis - This view displays data that has *financial instruments* as its main point of reference. The types of instrument-related data that can be viewed include indicative data, market data, accounting data, and relationship data (benchmarks, hedges, spreads, etc.).
2. Blotter View - This displays data that has *deals* as its main point of reference. The types of data that can be viewed for deals include transaction type, issue, price, quantity, counterparty, settle date, sales rep, etc.

3. Account View - This displays data that has a specific *account* as its main point of reference. The types of account data that can be viewed include legal entity (contact, address, etc.), accounting data (P&L, long position, short position, gross position, etc.), deal specifics (buys, sells, commissions, etc.), limits, settlement ladders, etc. Account data may be decomposed into sub-accounts and/or instrument sectors.

4. Risk View - This displays data that also has an account as its main point of reference. The types of data that can be viewed include position size, Risk Adjusted Position (RAP), hedge-equivalent position, etc. Risk data may also be decomposed into sub-accounts and/or instrument sectors.

5. Summary/Monitor View - This displays a variety of data types, and provides the ability to mix and match them in a spreadsheet-like matrix. Any RAPTR NT data can be displayed in a Summary View.

6. Deal Capture View - This allows users to enter, review, and modify deals. It is RAPT-NT's ticket entry screen.

7. Calculator Views - This allows users to perform ad hoc calculations such as finding forward settlement prices, carry costs, optimal strip/recon candidates, and spread values. It can also be used for "what-if" analyses and special analytic studies such as OAS and option valuations. Calculator types include: Constant Spread Calculator (calculates spread between Benchmark and dependent Instruments); Finance Trade (calculates cost of holding position); Butterfily (calculates price/yield relationship); and Option-Adjusted Spread (calculates values of fixed income options).

Finally, RAPTr-NT's architecture was specifically designed to extensible. This permits new applets to be seamlessly incorporated into the product. A well-defined API and Dynamic Link Libraries (DLLs) permit this to be easily accomplished by Kestrel or customers themselves.

RAPTr-NT uses a well defined, layered architecture that enables the product to be extended without affecting existing applications. Indeed, applications can manipulate new objects regardless of whether they were originally part of the product or were added later as part of a general system upgrade or through bespoke development. Descriptions of the object classes that are packaged in each layer of the system are as follows:

1. Phoenix Foundation (WC) Classes - These MFC-based classes are the support classes for RAPTr-NT's Trading Application and applets. They include:
   a. Window: for general MFC window-based classes;
   b. Grid: for general MFC grid-based classes;
   c. Dialog: for general MFC dialog-based classes.
2. Phoenix Applet (NEC) Classes - These MFC-based classes are the base classes for the RAPTr-NT Trading Application and applets.
3. Phoenix Foundation (ATL) Classes - These ATL-based classes are the base classes used within RAPTr-NT. They include:
   a. Packet: used for general data and object transport;
   b. Object Exchange: used for transferring state changes between objects and packets;
   c. Persistent: used for executing persistent semantics of creation, modification and deletion;
   d. Persistent Control: used for object location, querying and update management;

e. Object Mangers: used for management of transaction requests and object updates;

f. Object Filtering: used for filtering objects by type and state;

g. Defaults/Describe: used for storage of object/user settings;

h. Admin and Browser: used for object maintenance and object discovery;

i. Format: used for input conversion and output formatting.

4. Phoenix Business (ATL) Classes - These ATL-based classes are the base financial classes used within RAPTr-NT. They include:

a. Instrument: used for maintaining instrument indicative data;

b. Issuer: used for maintaining issuer information;

c. Sector: organizes instruments into rule-based collections;

d. Sector Link: provides a mechanism to create sector hierarchies;

e. Analytic: provides an interface and maintains state for instrument analytics;

f. Quote: provides an interface and maintains states for instrument pricing;

g. Deal: maintains transaction states;

h. Account: maintains descriptive information about accounts;

i. Account Link: provides a mechanism to create account hierarchies;

j. Legal Entity: maintains descriptive information about legal entities;

k. Stock Record: used to maintain position information by settlement date;

l. Cost Record: used to maintain position cost information by deal;

m.. P&L Record: used to maintain running P&L totals;

n. Currency Record: used to maintain cash positions;

o. Book: associates accounts, instruments, etc., while also functioning as a calculation, engine;

p. Book Work Unit: supplies the execution semantics required to process a unit of work;

q. Book Work Object: handles the processing work of individual object events;

r. Graph: maintains lists of Work Objects, sectors, and accounts;

s. Calendar: maintains holiday information per depository;

t. User: maintains descriptive information about users;

u. Organization: maintains descriptive information about organizations;

v. Organization User Link: links users and organizations,

w. Access Control List: provides access control for specific objects;

x. Series: maintains vectors of key data, for example, active issues, Strips, CPI Index, term structure rates, etc.

5. Kestrel (ATL) Classes. These ATL-based classes extend the Phoenix Classes, which results in the RAPTr-NT application.

They include classes that extend the Instrument, Quote, Analytic, etc. classes.

The RAPTr-NT Logical Model provides a flexible architecture that enables individual components to be added, modified, and removed from the product without impacting intervening components. This allows RAPTr-NT to be quickly and economically modified to meet changing business requirements and market conditions without sacrificing end user performance or functionality. Finally, the Model allows objects to be added to the system on the fly without adversely impacting existing applications or infrastructure components.

RAPTr-NT's architecture has been specifically designed to support a state-of-the-art front-office trading and investment management system that economically satisfies all critical business requirements. The most important of these are as follows:

- **Y2K Compliance.** Regulatory authorities are now demanding that dealing and investment management firms certify that their securities trading and processing systems are year 2000 compliant. In many cases, it is not economical to upgrade existing front-office solutions, nor is there sufficient time available to develop replacements from scratch. Kestrel's RAPTr-NT product is compliant, economical, and available today;
- **Straight-Through-Processing**. To reduce operating costs, securities firms need to automate the entire trading process from deal capture to settlement clearance and accounting. RAPTr-NT's advanced, industry-standard Middleware layer, enables firms to seamlessly link it via a guaranteed messaging mechanism with all manner of internal legacy and third-party

backoffice systems. The product also tightly integrates the workstation functions of traders and salespeople, which facilitates ticket matching, breakdown, and reconciliation operations;

- **Electronic Trading.** The movement toward electronic trading for liquid fixed-income securities is rapidly gaining momentum, as new liquidity networks such as *Trade Web, Trade Book* (Bloomberg), and *Instinet for Bonds* (Reuters) are being rolled out. RAPTr-NT has been designed to interface with all such networks and the Internet, thereby allowing both dealers and investors to take full advantage of the speed and efficiencies that electronic trading offers;
- **Global Trading and Risk Management.** With securities dealing now a world-wide, 24-hour business, firms require the ability to manage their global inventories. RAPTr-NT is the only commercially available system with the ability to provide a consistent, real-time, enterprise-wide view of their transactions, holdings, performance, and risk. Major institutions have spent many years and hundreds of millions of dollars tying to develop such technology with little success. Kestrel provides it out of the box;

ECU-Denominated Securities. Fixed-income securities denominated in the new European Currency Unit will soon be issued, and market participants will have to have a way to trade and manage their inventories of these instruments. RAPTr-NT can not only easily handle these securities, but it can also relate them to outstanding issues denominated in old Common Market

currencies.. The product will therefore allow individual traders to easily manage a completely heterogeneous mix of European fixed-income instruments;

In the exemplary embodiment, Server System 12 comprises a Sun E420R file server in conjunction with appropriate amount of supporting storage. For example, with between 1-5 users and less than 150 instruments per book could be configured with one file server with 2Gb RAM, 2x18.3Gb internal disk, CDROM, standard graphics, no monitor, Solaris 2.6 (May, 98). For 6-20 users and less than 500 Instruments (Securities) Per Book, 2 Sun file servers (one for the Oracle Database and one for the Application), 4Gb RAM (one per processor per machine), 2x18.3Gb internal disk, CDROM, standard graphics, no monitor, Solaris 2.6 (May, 98) would be suitable. For larger numbers, such as 21-35 users and less than 1500 Instruments (Securities) Per Book, 2 Sun file servers (one for the Oracle Database and one for the Application), 8Gb RAM (two per processor per machine), 2xl8.3Gb internal disk, CDROM, standard graphics, no monitor, Solaris 2.6 (May, 98) would be suitable. Finally, for 36 or more users and greater than 1500 Instruments (Securities) Per Book, 2 Sun E350OR file servers (one for the Oracle Database and one for the Application), 8Gb RAM (two per processor per machine), 2x18.3Gb internal disk, CDROM, standard graphics, no monitor, Solaris 2.6 (May, 98) would be suitable.

Also in the exemplary embodiment, workstation 14 may include an Intel Pentium 11 600 Mhz personal computer, 256Mb RAM (dedicated for RAPTr-NT more computing power is highly recommended if system

workstation will be used for other applications - i.e. MS Word, MS Excel, etc.), 4Gb internal disk (must have 500 Mb of dedicated space for RAPTr-NT application), high resolution fast graphics card W/ 4Mb RAM, 17" color monitor, CD-ROM, MS Windows NT 4.0 Service Pack 4 Operating System or higher, 3COM 515 10/100-Base-T Ethernet Adapter (or equivalent).

The Network connections of the present invention involve an Ascend Pipeline 50 ISDN router, or other equipment providing similar performance, with Ethernet Local Area Network with Category 5 wiring. Workstation 14 should have a Dedicated or "on demand" ISDN 2x64Kbs circuits or 56K dedicated circuit.

While this invention has been described as having an exemplary design, the present invention may be further modified within the spirit and scope of this disclosure. This application is therefore intended to cover any variations, uses, or adaptations of the invention using its general principles. Further, this application is intended to cover such departures from the present disclosure as come within known or customary practice in the art to which this invention pertains.

WHAT IS CLAIMED IS:

1. A trading system used with a data processing system that manages and trades fixed interest securities including transacting the purchase and sale of at least one security having a set of characteristic data, where the data processing system is operated by a plurality of trading participants and transmits information relating to the securities characteristic data, with a workstation that presents to a participant information about pending market conditions as they relate to the security characteristic data of at least one security, the workstation being in communication with a server (12) which provides event information to the workstation, the workstation including at least one securities evaluation software application (SE) which references an object having time sensitive security characteristic data, the workstation characterized by persistent control software (PC) having instructions for enabling said workstation to update said object having said time sensitive security characteristic data when external event information is received and said external event information is relevant to said time sensitive securities data.

2. The trading system of Claim 1 characterized in that the workstation further includes object manager software (OMI).

3. The trading system of Claim 2 characterized in that the object manager software includes update software (TOM) for modifying data in an object in response to external event information.

4 The trading system of Claim 1 characterized in that the persistent control includes a database (TSA) referencing a plurality of objects.

5. The trading system of Claim 4 characterized in that the database includes state information relating to said plurality of objects.

6. The trading system of Claim 1 characterized in that the server includes a messaging system (MI) allowing at least one of said plurality of objects to be located on said server and be referenced by said at least one securities evaluation software application.

7. The trading system of Claim 1 characterized in that the server is configured to interact with a plurality of ECNs (10).

8. The trading system of Claim 8 characterized in that the workstation includes virtual market software (20) for coordinating said plurality of ECNs for said securities evaluation software application.

9. In a computer network, a method of managing and trading fixed interest securities including transacting the purchase and sale of at least one security having a set of characteristic data, operated by a plurality of trading participants and transmits information relating to the securities characteristic data, with at least one trading participant having a workstation (14) with a securities evaluation software application (SE), said method comprising the steps of providing a securities evaluation software application which references an object having time sensitive security characteristic data which is characterized by providing persistent control software (PC) for updating the object having the time sensitive security characteristic data, preventing access to the securities evaluation software application except through the persistent control software, and receiving external event information and the persistent control determining whether the external event information is relevant to said time sensitive securities data.

10. The method of Claim 9 characterized by the step of maintaining a database of objects (TSA) including object state information, and said determining step is made in reference to the object state information.

11. The method of Claim 9 characterized by the step of coupling the securities evaluation software application to a plurality of ECNs (10).

12. The method of Claim 11 characterized by the step of providing virtual market software (20) which coordinates the plurality of ECNs for the securities evaluation software application.

13. The method of Claim 9 characterized by the step of providing object manager software (OMI), and further comprising the step of said object manager software updating modifying data in an object in response to external event information.

14. A machine-readable program storage device for storing encoded instructions for a method of managing and trading fixed interest securities including transacting the purchase and sale of at least one security having a set of characteristic data, wherein said data processing system is operated by a plurality of trading participants and transmits information relating to the securities characteristic data, with at least one trading participant having a workstation (14) with a securities evaluation software application (SE), said method comprising the steps of providing a securities evaluation software application which references an object having time sensitive security characteristic data, characterized by providing persistent control software (PC) for updating the object having the time sensitive security characteristic data, preventing access to the securities evaluation software application except through the persistent control software, and receiving external event information and the persistent control determining whether the external event information is relevant to said time sensitive securities data.

15. The machine-readable program storage device of Claim 14 wherein said method is characterized by the step of maintaining a database of objects (TSA) including object state information, and said determining step is made in reference to the object state information.

16. The machine-readable program storage device of Claim 14 wherein said method characterized by the step of coupling the securities evaluation software application to a plurality of ECNs (10).

17. The machine-readable program storage device of Claim 16 wherein said method characterized by the step of providing virtual market software (20) which coordinates the plurality of ECNs for the securities evaluation software application.

18. The machine-readable program storage device of Claim 14 wherein said method characterized by the step of providing object manager software (OMI), and further comprising the step of said object manager software updating modifying data in an object in response to external event information.

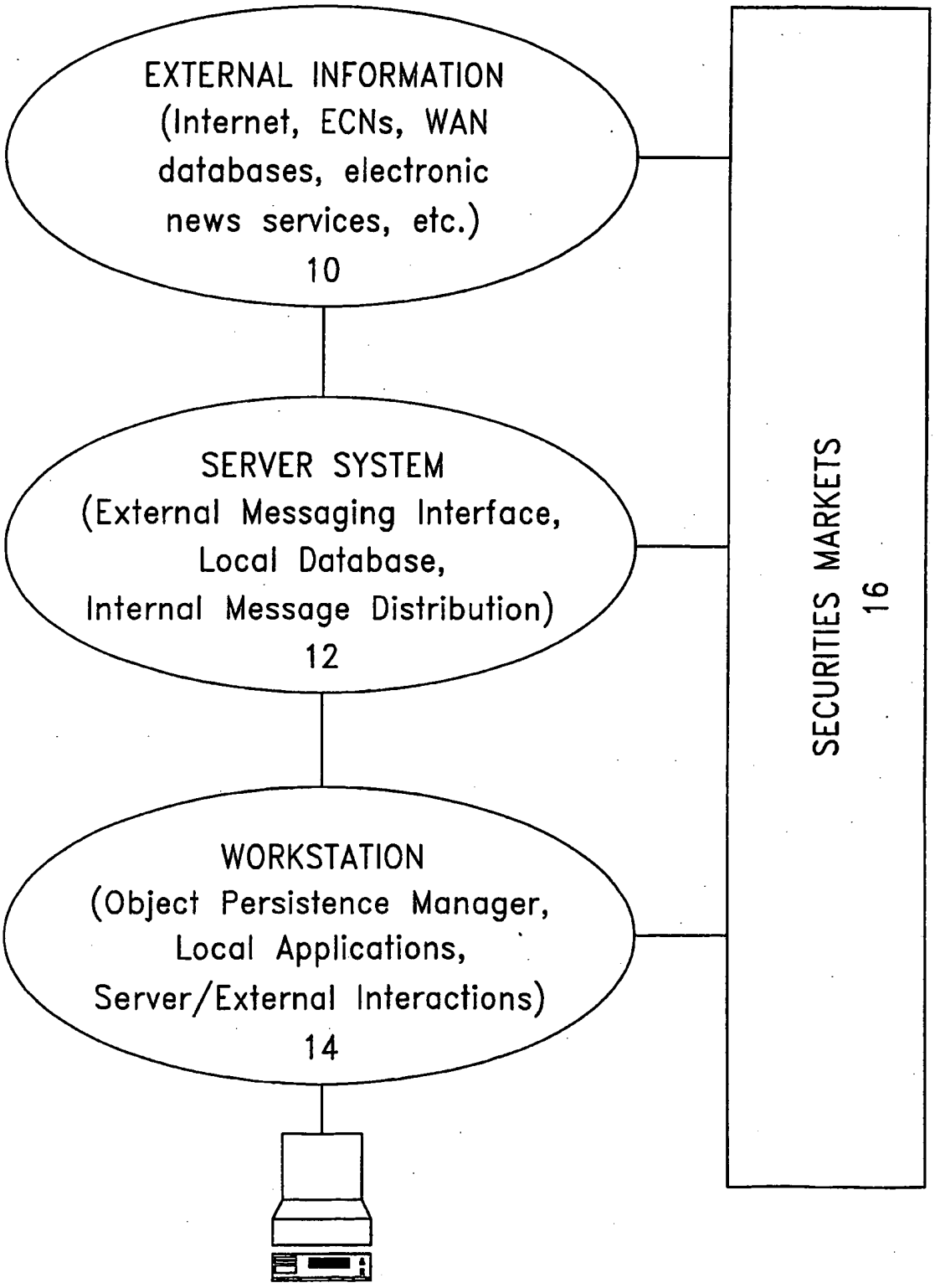

EXTERNAL INFORMATION
(Internet, ECNs, WAN
databases, electronic
news services, etc.)
10
SERVER SYSTEM
(External Messaging Interface,
Local Database,
Internal Message Distribution)
12
WORKSTATION
(Object Persistence Manager,
Local Applications,
Server/External Interactions)
14
SECURITIES MARKETS
16
USER WORKSTATION DISPLAY
18

Fig. 1

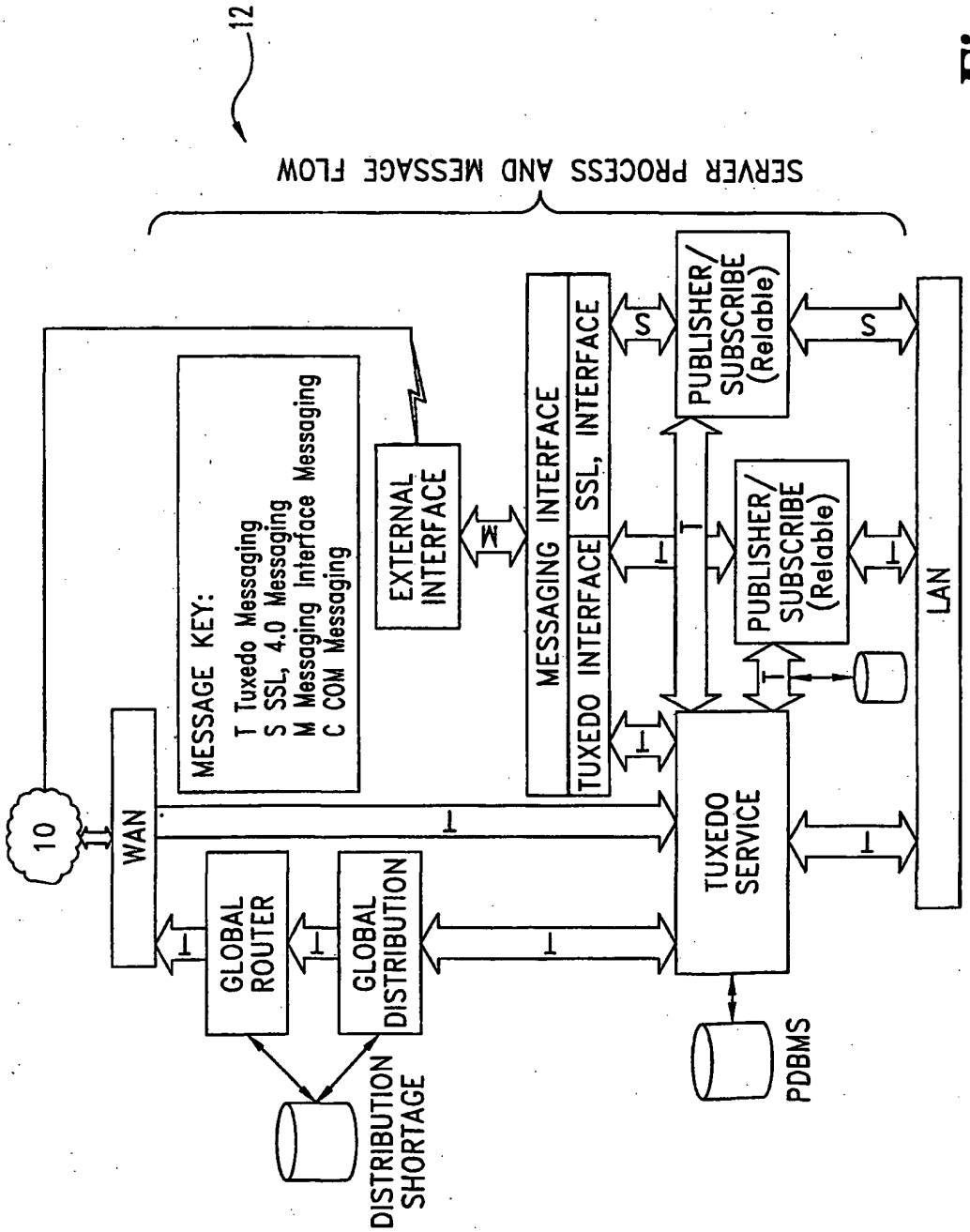

12
SERVER PROCESS AND MESSAGE FLOW
MESSAGE KEY:
T Tuxedo Messaging
S SSL, 4.0 Messaging
M Messaging Interface Messaging
C COM Messaging
EXTERNAL INTERFACE
M
MESSAGING INTERFACE
TUXEDO INTERFACE
SSL, INTERFACE
PUBLISHER/ SUBSCRIBE (Relable)
PUBLISHER/ SUBSCRIBE (Relable)
S
S
T
T
LAN
10
WAN
GLOBAL ROUTER
GLOBAL DISTRIBUTION
TUXEDO SERVICE
PDBMS
DISTRIBUTION SHORTAGE
MATCH FIG. 2B

Fig. 2A

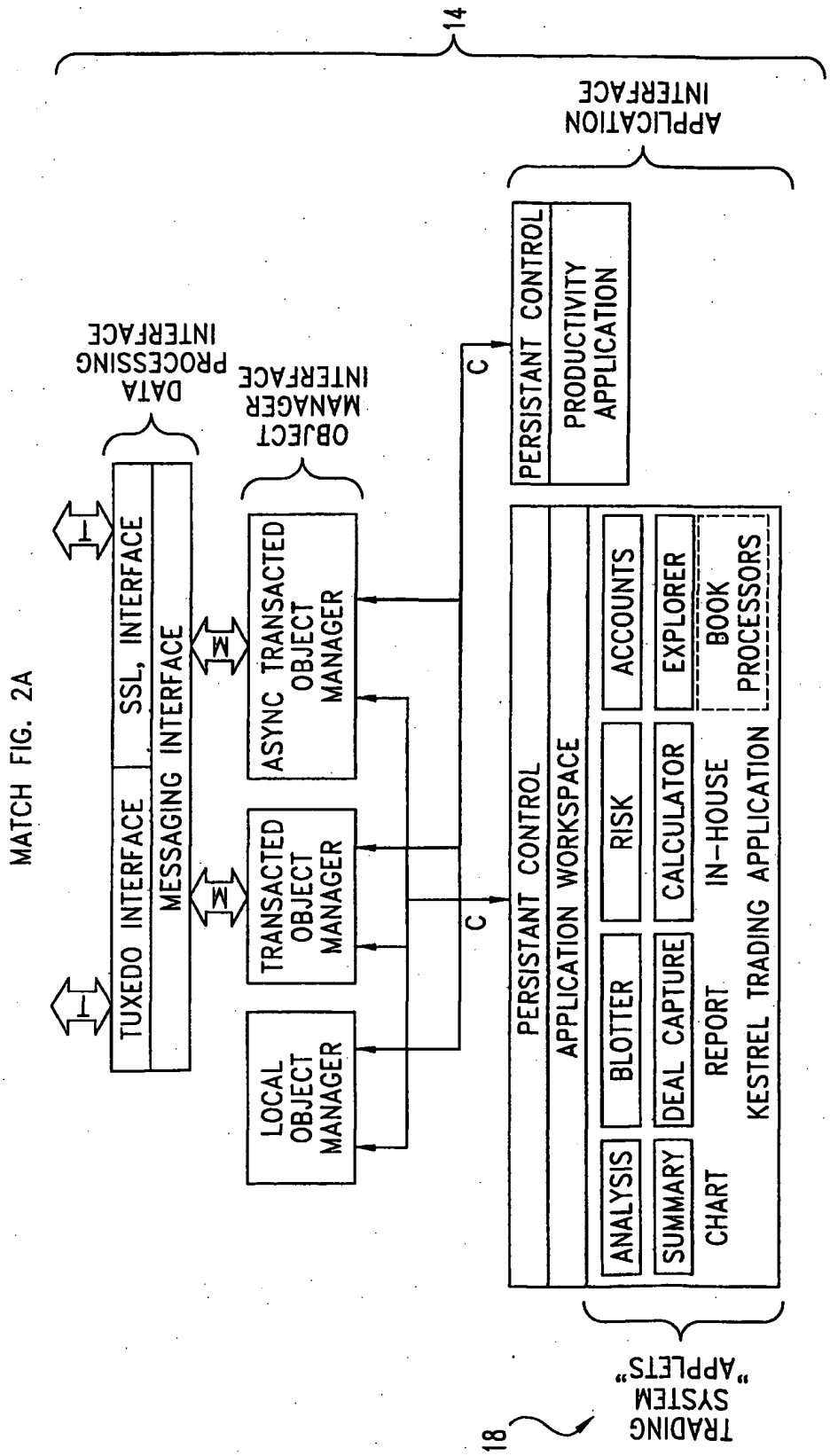

14
APPLICATION INTERFACE
PERSISTANT CONTROL
PRODUCTIVITY APPLICATION
DATA PROCESSING INTERFACE
OBJECT MANAGER INTERFACE
MATCH FIG. 2A
T
SSL, INTERFACE
TUXEDO INTERFACE
MESSAGING INTERFACE
M
ASYNC TRANSACTED OBJECT MANAGER
TRANSACTED OBJECT MANAGER
LOCAL OBJECT MANAGER
C
PERSISTANT CONTROL
APPLICATION WORKSPACE
ANALYSIS
SUMMARY
CHART
BLOTTER
DEAL CAPTURE
REPORT
RISK
CALCULATOR
IN-HOUSE
ACCOUNTS
EXPLORER
BOOK PROCESSORS
KESTREL TRADING APPLICATION
TRADING SYSTEM "APPLETS"
18

Fig. 2B

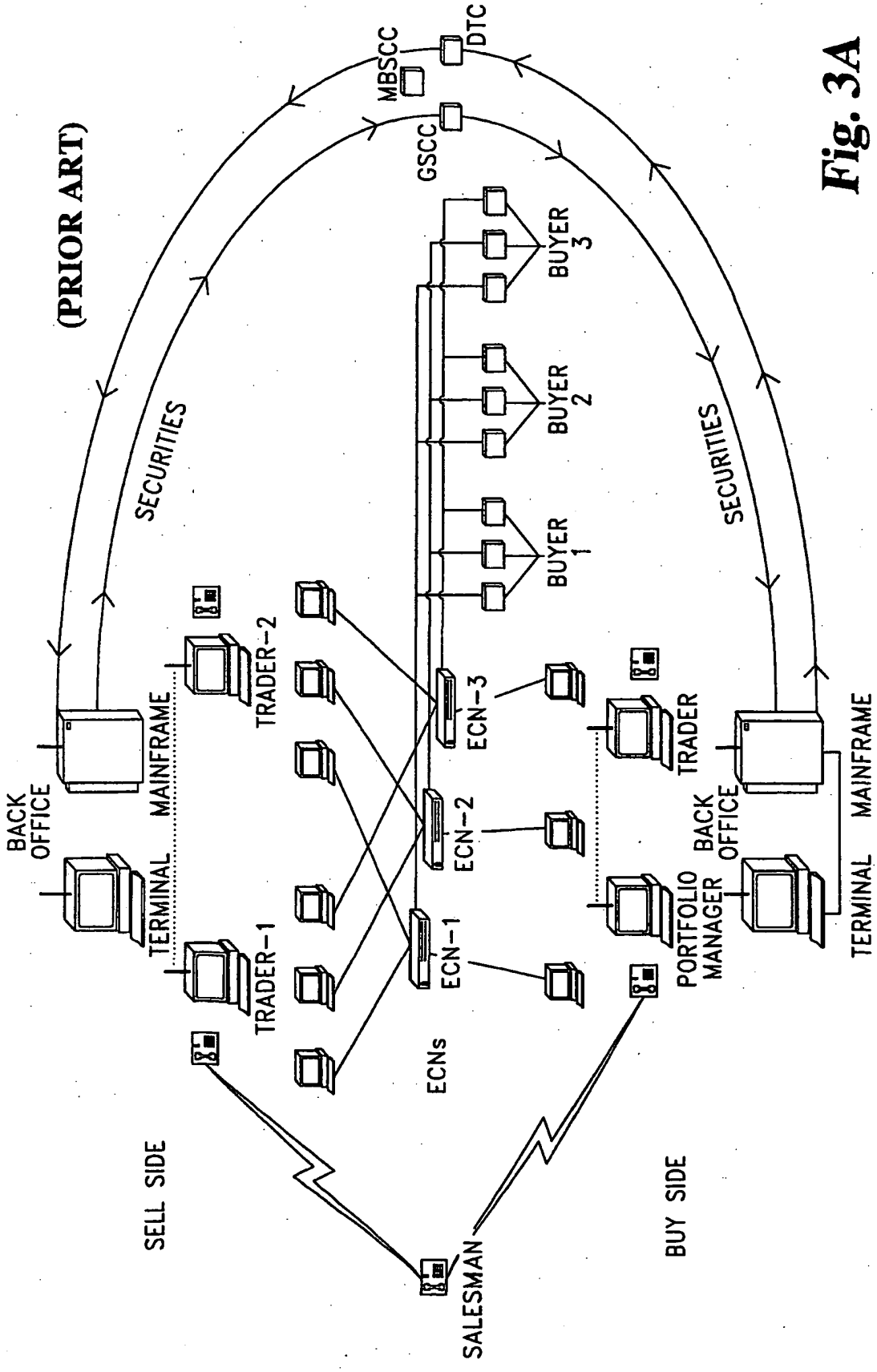

(PRIOR ART)
MBSCC
DTC
GSCC
SECURITIES
SECURITIES
BUYER 3
BUYER 2
BUYER 1
BACK OFFICE
TERMINAL
MAINFRAME
TRADER-2
TRADER-1
ECN-3
ECN-2
ECN-1
ECNs
TRADER
PORTFOLIO MANAGER
BACK OFFICE
TERMINAL
MAINFRAME
SELL SIDE
BUY SIDE
SALESMAN

Fig. 3A

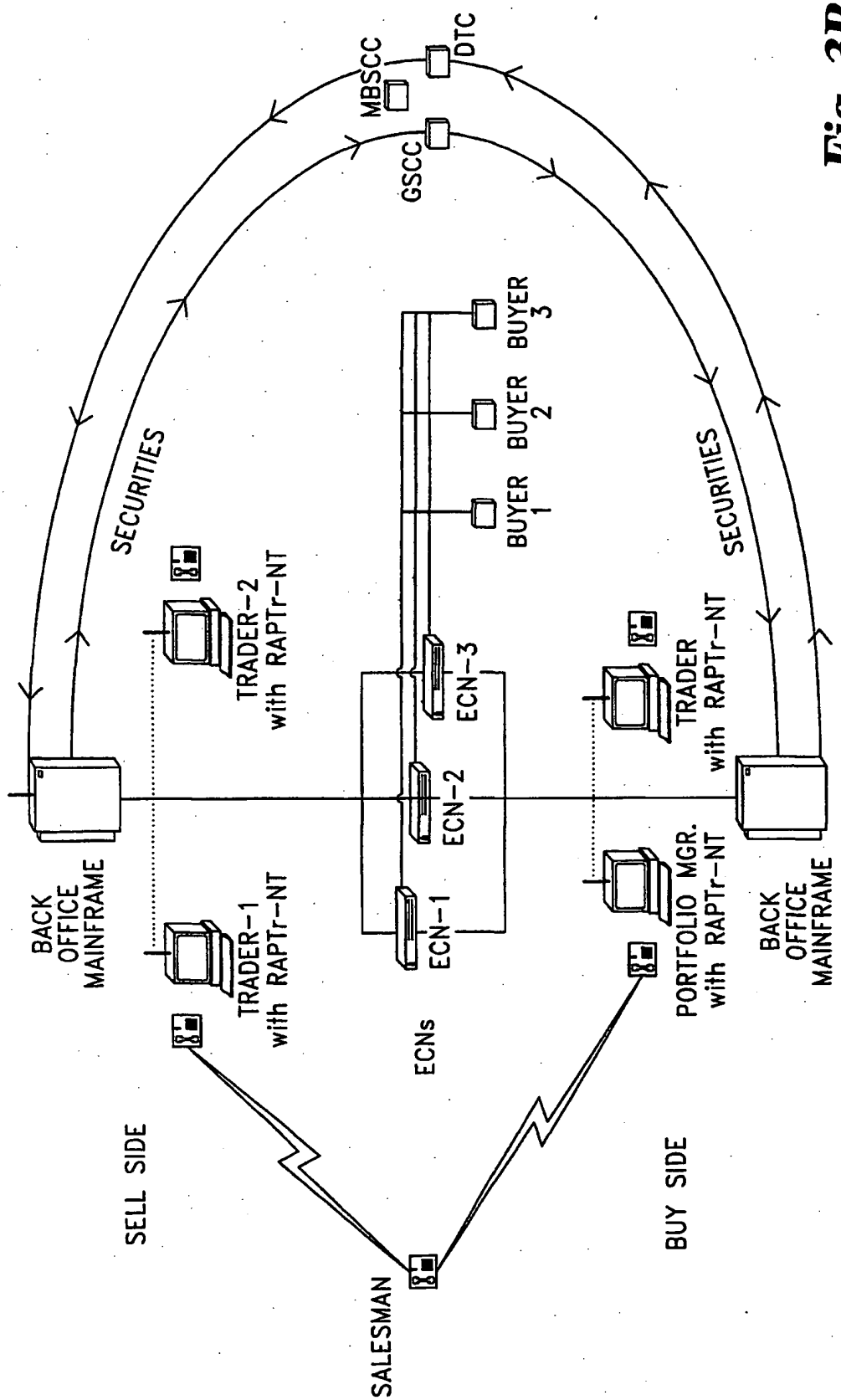

MBSCC
DTC
GSCC
SECURITIES
SECURITIES
BUYER 1
BUYER 2
BUYER 3
TRADER-2 with RAPTr-NT
TRADER with RAPTr-NT
ECN-3
ECN-2
ECN-1
ECNs
BACK OFFICE MAINFRAME
BACK OFFICE MAINFRAME
TRADER-1 with RAPTr-NT
PORTFOLIO MGR. with RAPTr-NT
SELL SIDE
BUY SIDE
SALESMAN

Fig. 3B